Today’s Agenda: All about Learning

- Unsupervised Learning
  - Sparse Coding
  - Predictive Coding

- Supervised learning
  - Perceptrons and Backpropagation

- Reinforcement Learning
  - TD and Actor-Critic learning
Recall from Last Time: Linear Generative Model

Suppose input $u$ was generated by a linear superposition of causes $v_1$, $v_2$, ..., $v_k$ with basis vectors (or “features”) $g_i$

$$u = \sum_i g_i v_i + noise = Gv + n$$
(Assume noise is Gaussian white noise with mean zero)

Bayesian approach

- Find $v$ and $G$ that maximize posterior:
  $$p[v \mid u; G] = k \cdot p[u \mid v; G]p[v; G]$$

- Equivalently, find $v$ and $G$ that maximize log posterior:
  $$F(v, G) = \log p[u \mid v; G] + \log p[v; G] + \log k$$

  \[
  u = Gv + n \\
  \log \text{of Gaussian} \\
  \log N(u; Gv, I) \\
  = -\frac{1}{2} (u - Gv)^T (u - Gv) + C
  \]

  If $v_a$ independent
  $$p[v; G] = \prod_a p[v_a; G]$$
  $$\log p[v; G] = \sum_a \log p[v_a; G]$$

  Prior for individual causes (what should this be?)
What do we know about the causes \( \mathbf{v} \)?

- Idea: Causes independent: only a few of them will be active for any input
  - \( v_a \) will be 0 most of the time but high for a few inputs
  - Suggests a sparse distribution for \( p[v_a; G] \): peak at 0 but with heavy tail (also called super-Gaussian distribution)

Examples of Prior Distributions for Causes

Possible prior distributions

Log prior

\[
p[\mathbf{v}; G] = c \cdot \prod_a \exp(g(v_a))
\]

\[
\log p[\mathbf{v}; G] = \sum_a g(v_a) + c
\]
Finding the optimal \( v \) and \( G \)

- Want to maximize:

\[
F(v, G) = \log p[u \mid v; G] + \log p[v; G] + \log k
\]

\[
= -\frac{1}{2} (u - Gv)^T (u - Gv) + \sum_a g(v_a) + K
\]

- Alternate between two steps:
  - Maximize \( F \) with respect to \( v \) keeping \( G \) fixed
    - How?
  - Maximize \( F \) with respect to \( G \), given the \( v \) above
    - How?

Estimating the causes \( v \) for a given input

Gradient ascent

\[
\frac{dv}{dt} \propto \frac{dF}{dv} = G^T (u - Gv) + g'(v)
\]

- Derivative of \( g \)
- Reconstruction (prediction) of \( u \)
- Firing rate dynamics (Recurrent network)
- Error
- Sparseness constraint
Sparse Coding Network for Estimating $v$

$$\tau \frac{dv}{dt} = G^T (u - Gv) + g'(v)$$

Corrcted Estimate

[Suggests a role for feedback pathways in the cortex (Rao & Ballard, 1999)]

Learning the Synaptic Weights $G$

$$\frac{dG}{dt} \propto \frac{dF}{dG} = (u - Gv)v^T$$

Gradient ascent

Learning rule

$$\tau G \frac{dG}{dt} = (u - Gv)v^T$$  \{Hebbian! (similar to Oja’s rule)\}
Result: Learning $G$ for Natural Images

Each square is a column $g_i$ of $G$ (obtained by collapsing rows of the square into a vector).

Almost all the $g_i$ represent local edge features.

Any image patch $u$ can be expressed as:

$$u = \sum_i g_i v_i = Gv$$

Sparse Coding Network is a special case of Predictive Coding Networks

(Olshausen & Field, 1996)
Predictive Coding Model of Visual Cortex


Predictive coding model explains contextual effects

Monkey Primary Visual Cortex

Increased activity for non-homogenous input interpreted as prediction error (i.e., anomalous input): center is not predicted by surrounding context.

(Zipser et al., *J. Neurosci.*, 1996)
Natural Images as a Source of Contextual Effects

What if your data comes with not just inputs but also outputs?

Enter…Supervised Learning
Supervised Learning

Two Primary Tasks

1. **Classification**
   - Inputs \( u_1, u_2, \ldots \) and discrete classes \( C_1, C_2, \ldots, C_k \)
   - Training examples: \((u_1, C_2), (u_2, C_7), \text{ etc.}\)
   - Learn the mapping from an arbitrary input to its class
   - Example: Inputs = images, output classes = face, not a face

2. **Regression**
   - Inputs \( u_1, u_2, \ldots \) and continuous outputs \( v_1, v_2, \ldots \)
   - Training examples: (input, desired output) pairs
   - Learn to map an arbitrary input to its corresponding output
   - Example: Highway driving
     - Input = road image, output = steering angle

The Classification Problem

- • denotes output of +1 (faces)
- ○ denotes output of -1 (other)

Idea: Find a separating hyperplane (line in this case)
Neurons as Classifiers: The “Perceptron”

Artificial neuron:
- m binary inputs (-1 or 1) and 1 output (-1 or 1)
- Synaptic weights $w_{ij}$
- Threshold $\mu_i$

\[ v_i = \Theta(\sum_j w_{ij} u_j - \mu_i) \]

$\Theta(x) = +1$ if $x \geq 0$ and -1 if $x < 0$

What does a Perceptron compute?

Consider a single-layer perceptron
- Weighted sum forms a *linear hyperplane* (line, plane, ...)

\[ \sum_j w_{ij} u_j - \mu_i = 0 \]

- Everything *on one side* of hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)
- *Any function that is linearly separable can be computed by a perceptron*
Linear Separability

Example: AND function is linearly separable
⇒ a AND b = 1 if and only if a = 1 and b = 1

Linear hyperplane

Perceptron for AND

What about the XOR function?

<table>
<thead>
<tr>
<th>$u_1$</th>
<th>$u_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>+1</td>
</tr>
</tbody>
</table>

Can a straight line separate the +1 outputs from the -1 outputs?
Multilayer Perceptrons

- Removes limitations of single-layer networks
  - Can solve XOR
- An example of a two-layer perceptron that computes XOR
  
  ![Diagram](image)

- Output is +1 if and only if \( x + y + 2\Theta(- x - y - 1.5) > -1 \)
  (Inputs x and y can be +1 or -1)

What if you want to approximate a *continuous* function (i.e., regression)?

Can a network learn to drive?
Example Network

Steering angle

Desired Output:
\( \mathbf{d} = [d_1, d_2, \ldots, d_{30}] \)

Current image

Input \( \mathbf{u} = [u_1, u_2, \ldots, u_{960}] = \text{image pixels} \)

Sigmoid Networks

Output \( \mathbf{v} = g(\mathbf{w}^T \mathbf{u}) = g(\sum_i w_i u_i) \)

Sigmoid output function:
\[
g(a) = \frac{1}{1 + e^{-\beta a}}
\]

Sigmoid is a non-linear “squashing” function: Squashes input to be between 0 and 1. Parameter \( \beta \) controls the slope.
Multilayer Sigmoid Networks

\[ v_i = g(\sum_j W_{ji} g(\sum_k w_k u_k)) \]

Output \( v = (v_1 \ v_2 \ \ldots \ v_J)^T \); Desired = \( d \)

Input \( u = (u_1 \ u_2 \ \ldots \ u_K)^T \)

How do we learn these weights?

Backpropagation Learning: Uppermost layer

Minimize output error:

\[ E(W, w) = \frac{1}{2} \sum_i (d_i - v_i)^2 \]

Learning rule for hidden-output weights \( W \):

\[ W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \quad \{ \text{gradient descent} \} \]

\[ \frac{dE}{dW_{ji}} = -(d_i - v_i) g'(\sum_j W_{ji} x_j) x_j \quad \{ \text{delta rule} \} \]
Backpropagation: Inner layer (chain rule)

Minimize output error:

$$E(W, w) = \frac{1}{2} \sum_i (d_i - v_i)^2$$

$$v^m_i = g(\sum_j W_{ji}x_j)$$

$$x^m_j = g(\sum_k w_{jk}u^m_k)$$

Learning rule for input-hidden weights $w$:

$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}}$$

But:

$$dE \over dw_{kj} = dE \over dx_j \cdot dx_j \over dw_{kj} \{\text{chain rule}\}$$

$$dE \over dw_{kj} = \left[-\sum_{m,i} (d^m_i - v^m_i)g'(\sum_j W_{ji}x^m_j)W_{ji}\right] \cdot g'(\sum_k w_{jk}u^m_k)u^m_k$$

Demos: Pole Balancing and Backing up a Truck

(courtesy of Keith Grochow, CSE 599)

- Neural network learns to balance a pole on a cart
  - System:
    - 4 state variables: $x_{\text{cart}}, v_{\text{cart}}, \theta_{\text{pole}}, v_{\text{pole}}$
    - 1 input: Force on cart
  - Backprop Network:
    - Input: State variables
    - Output: New force on cart
- NN learns to back a truck into a loading dock
  - System (Nyugen and Widrow, 1989):
    - State variables: $x_{\text{cab}}, y_{\text{cab}}, \theta_{\text{cab}}$
    - 1 input: new $\theta_{\text{steering}}$
  - Backprop Network:
    - Input: State variables
    - Output: Steering angle $\theta_{\text{steering}}$
Humans (and animals in general) don’t get exact supervisory signals (commands for muscles) for learning to talk, walk, ride a bicycle, play the piano, drive, etc.

We learn by trial-and-error (with hints from others)

Might get “rewards and punishments” along the way

**Enter…Reinforcement Learning**

---

The Reinforcement Learning “Agent”

![Diagram of Reinforcement Learning](image)
Early Results: Pavlov and his Dog

- Classical (Pavlovian) conditioning experiments
- **Training**: Bell $\rightarrow$ Food
- **After**: Bell $\rightarrow$ Salivate
- Conditioned stimulus (bell) predicts future reward (food)

(http://employees.csbsju.edu/tcreed/pb/pdoganim.html)

Predicting Delayed Rewards

- Reward is typically delivered at the end (when you know whether you succeeded or not)
- Time: $0 \leq t \leq T$ with stimulus $u(t)$ and reward $r(t)$ at each time step $t$ (Note: $r(t)$ can be zero at some time points)
- Key Idea: Make the output $v(t)$ predict total expected future reward starting from time $t$

$$v(t) \approx \left( \sum_{\tau=0}^{T-t} r(t+\tau) \right)$$
Learning to Predict Delayed Rewards

- Use a set of modifiable weights \( w(t) \) and *predict based on all past stimuli* \( u(t) \):

\[
v(t) = \sum_{\tau=0}^{T} w(\tau) u(t - \tau)
\]

- Would like to find the weights (or filter) \( w(\tau) \) that minimize:

\[
\left( \sum_{\tau=0}^{T-t} r(t + \tau) - v(t) \right)^2
\]

(Can we minimize this using gradient descent and delta rule?)

Yes, BUT…not yet available are the future rewards

---

Temporal Difference (TD) Learning

- **Key Idea:** Rewrite squared error to get rid of future terms:

\[
\left( \sum_{\tau=0}^{T-t} r(t + \tau) - v(t) \right)^2 = \left( r(t) + \sum_{\tau=0}^{T-t-1} r(t + 1 + \tau) - v(t) \right)^2 
\]

\[
\approx \left( r(t) + v(t + 1) - v(t) \right)^2
\]

Minimize this using gradient descent!

- **Temporal Difference (TD) Learning:**

\[
w(\tau) \rightarrow w(\tau) + \delta \left[ r(t) + v(t + 1) - v(t) \right] u(t - \tau)
\]

Expected future reward  
Prediction
Predicting Delayed Reward: TD Learning

Stimulus at $t = 100$ and reward at $t = 200$

Prediction error $\delta$ for each time step (over many trials)

Possible Reward Prediction Error Signal in the Primate Brain

Dopaminergic cells in Ventral Tegmental Area (VTA)

Reward Prediction error? $[r(t) + v(t+1) - v(t)]$

Before Training

After Training

No error

$[0 + v(t+1) - v(t)]$

$[r(t) + v(t+1) - v(t)] \approx 0$
More Evidence for Prediction Error Signals

Dopaminergic cells in VTA

reward

no reward

$\text{Negative error}$

$\begin{align*}
  r(t) &= 0, \nu(t + 1) = 0 \\
  [r(t) + \nu(t + 1) - \nu(t)] &= -\nu(t)
\end{align*}$

That’s great, but how does all that math help me get food in a maze?
Selecting Actions when Reward is Delayed

States: A, B, or C
Possible actions at any state: Left (L) or Right (R)

If you randomly choose to go L or R (random “policy”), what is the expected value $v$ of each state?

Policy Evaluation

For random policy:

- $v(B) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 5 = 2.5$
- $v(C) = \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 0 = 1$
- $v(A) = \frac{1}{2} \cdot v(B) + \frac{1}{2} \cdot v(C) = 1.75$

Can learn value of locations using TD learning:

Let value of location $v(u) =$ weight $w(u)$

$w(u) \rightarrow w(u) + \varepsilon \left[ r_u + v(u') - v(u) \right]$
Maze Value Learning for Random Policy

Once I know the values, I can pick the action that leads to the higher valued state!

(For all three, $\varepsilon = 0.5$)

Selecting Actions based on Values

Values act as surrogate immediate rewards $\rightarrow$ Locally optimal choice leads to globally optimal policy (for “Markov” environments)
Related to Dynamic Programming in CS (see appendix in text)
Actor-Critic Learning

- Two separate components: Actor (maintains policy) and Critic (maintains value of each state)

1. **Critic Learning ("Policy Evaluation"):**
   - Value of state \( u = v(u) = w(u) \)
   - \( w(u) \rightarrow w(u) + \varepsilon [r_a(u) + v(u') - v(u)] \) (same as TD rule)

2. **Actor Learning ("Policy Improvement"):**
   - \( P(a; u) = \frac{\exp(\beta Q_a(u))}{\sum_b \exp(\beta Q_b(u))} \)
   - Use this to select an action \( a \) at state \( u \)
   - For all \( a' \):
     - \( Q_a(u) \rightarrow Q_a(u) + \varepsilon [r_a(u) + v(u') - v(u)](\delta_{aa'} - P(a'; u)) \)

3. **Interleave 1 and 2**

---

Actor-Critic Learning in the Maze Task

![Maze Task Diagram]

Probability of going Left at a location

\( P[L; u] \)

- \( u = A \)
- \( u = B \)
- \( u = C \)
Demo of Reinforcement Learning in a Robot
(from http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robodemo.html )

Things to do:

Finish homework 3
Work on group project

Thanks, dopamine!